



Multi-core is Here! But How Do You Resolve Data Bottlenecks in PC Games? hint: it's all about *locality*

Michael Wall

Principal Member of Technical Staff, Advanced Micro Devices, Inc.
Game Developers Conference

February, 2008

Session Prerequisites

- Working knowledge of C/C++
- Basic understanding of microprocessor concepts
- Interest in making software run faster

Session Objectives and Agenda

- “Barcelona”: AMD’s new CPU architecture
 - The new AMD Opteron™ Processor family member
 - Also a new desktop CPU, the AMD Phenom™ Processor
 - Native quad-core microprocessor
 - Enhanced cache and memory systems
 - Improved memory controller
 - Level 3 cache (and L1, L2)
 - Hardware data prefetchers
- General cache/memory software optimization techniques
 - Data strategies to maximize memory performance
 - Data layout to maximize cache efficiency
 - Explicit cache control
 - NUMA
 - AMD CodeAnalyst profiler

Key features of the new processor family

Native quad-core upgrade for 2007

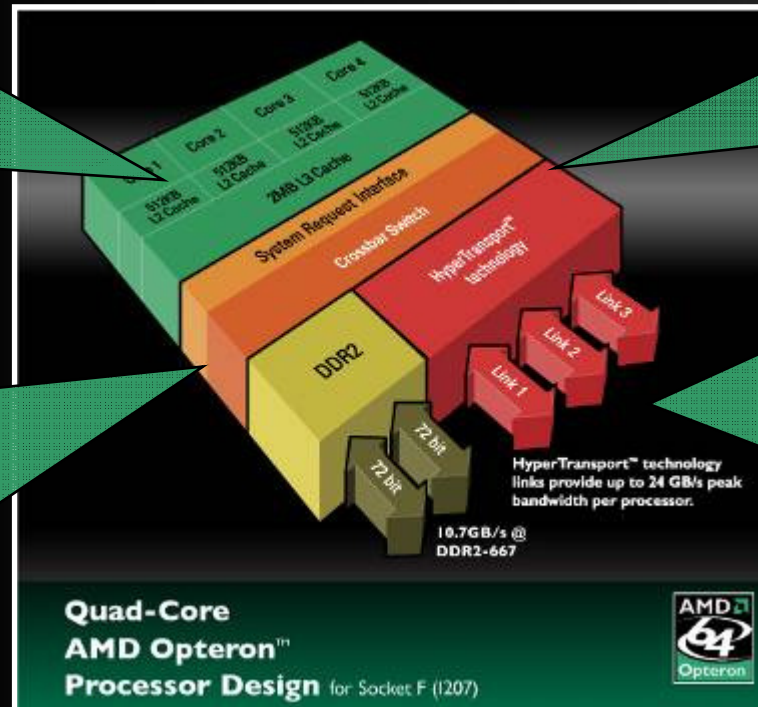
Native Quad-Core Processor

To increase performance-per-watt efficiencies using the same Thermal Design Power.

Advanced Process Technology

65nm Silicon-on-Insulator Process

Fast transistors with low power leakage to reduce power and heat.



Platform Compatibility

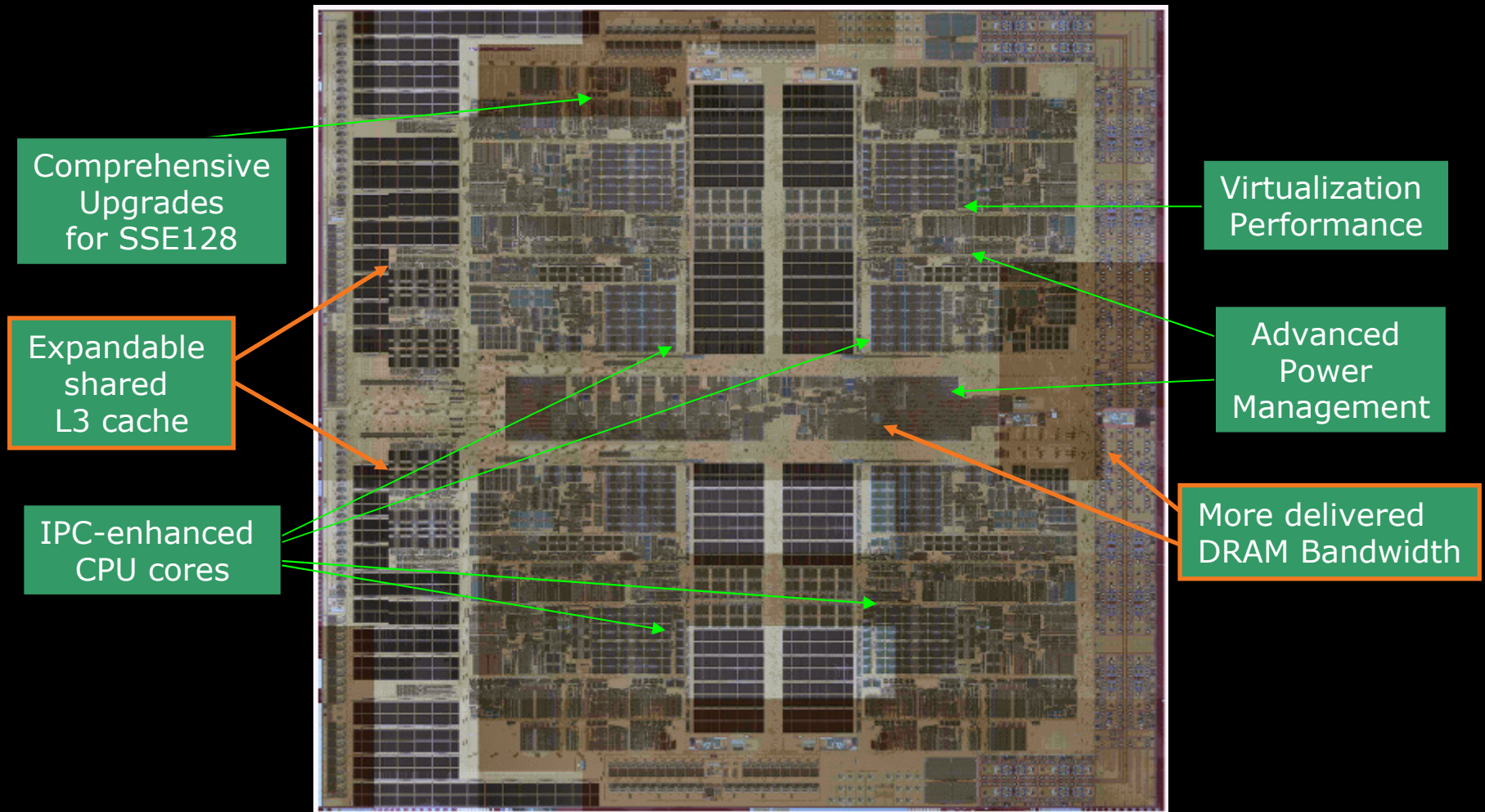
Socket and thermal compatible with "Socket F".

Direct Connect Architecture

- Integrated memory controller designed for reduced memory latency and increased performance
 - Memory directly connected
- Provides fast CPU-to-CPU communication
 - CPUs directly connected
- Glueless SMP up to 8 sockets

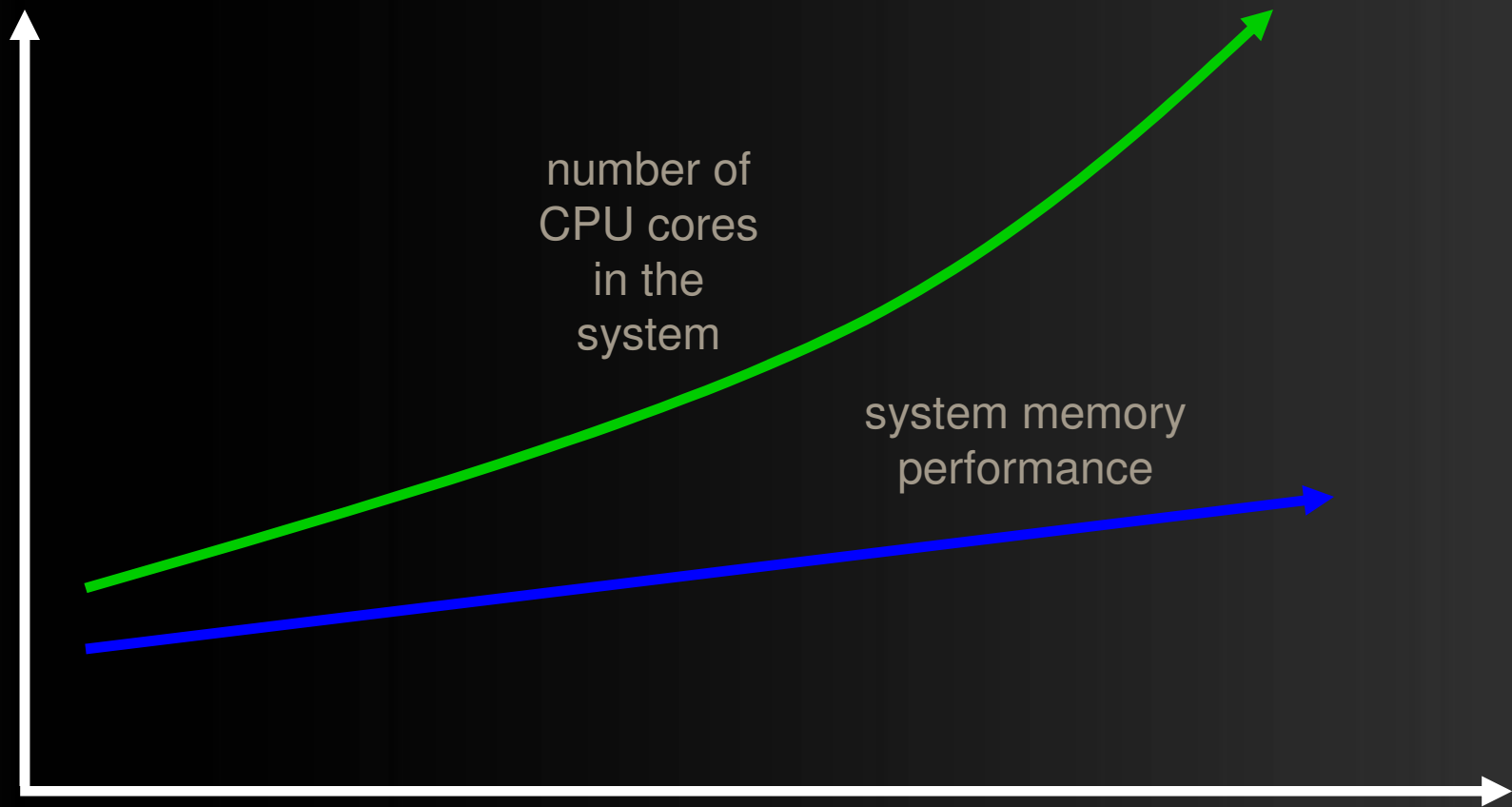
A Closer Look: AMD's New Processor

special focus on enhancements for cache and memory



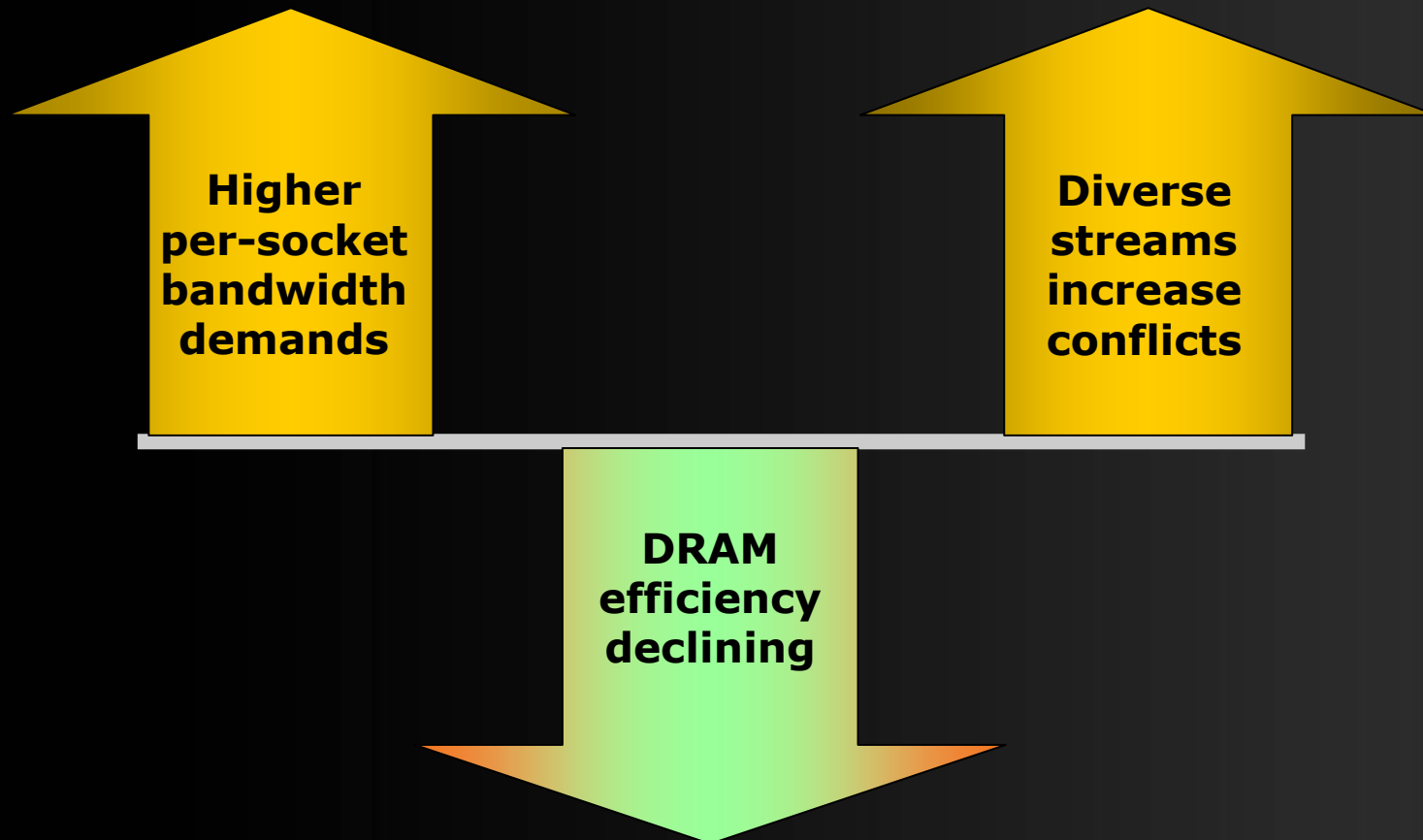
Another one of *those* graphs

CPU (cores) are increasing faster than DRAM



Trends in DRAM bandwidth

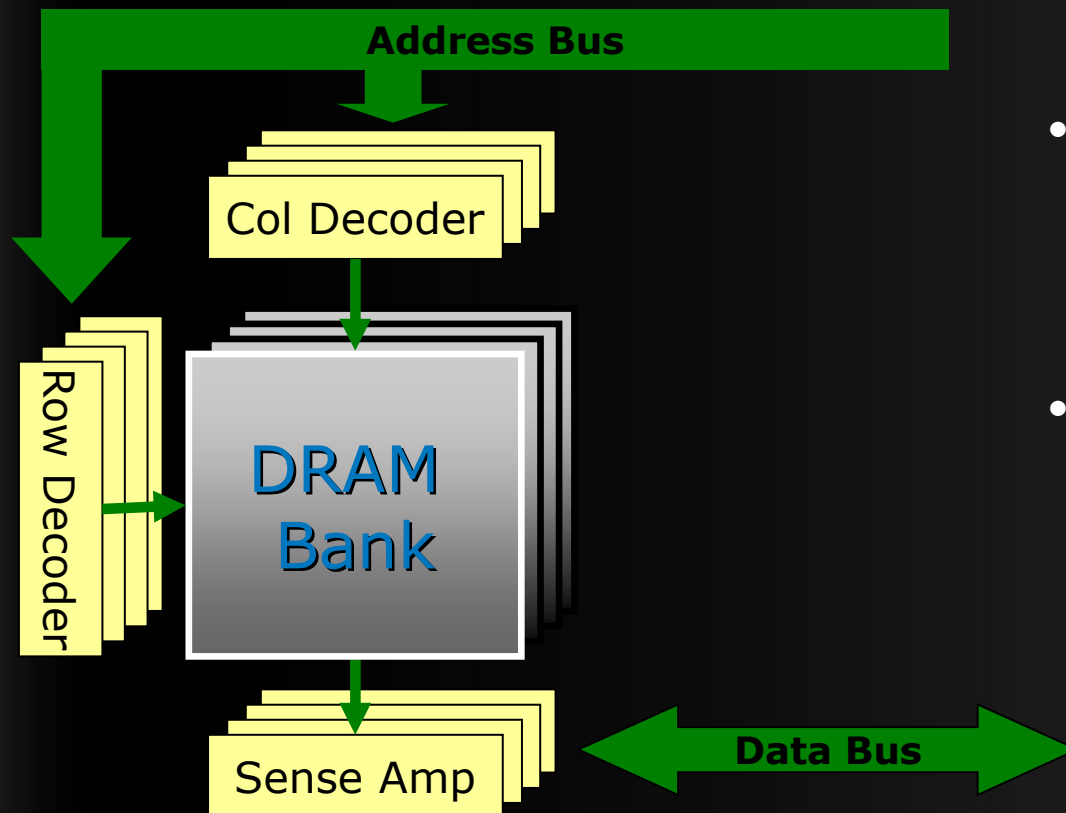
Improved Efficiency is the Answer



The CPU must improve *delivered* DRAM bandwidth

DRAM Basics

Dynamic RAM access is fairly complex



- Complex access protocol:
 - ACT to load row into sense amp
 - READ column from sense amp
 - PRECHARGE to reset sense amp
- Efficient Access Requires:
 - Access different banks
 - 4-8 banks/chip
 - 1-4 chips/channel
 - Column locality

Cmd:	ACT	ACT	ACT	NOP	NOP	RD	NOP	NOP	NOP	RD	PRE	NOP	NOP	RD	NOP	ACT	NOP	RD	NOP	NOP	NOP	RD	PRE	NOP	NOP	NOP	PRE	NOP	NOP
Data:											D0D1	D2D3	D4D5	D6D7	D0D1	D2D3	D4D5	D6D7	D0D1	D2D3	D4D5	D6D7	D0D1	D2D3	D4D5	D6D7	D0D1	D2D3	D4D5

Delivering more DRAM bandwidth

enhancements in AMD Barcelona processor

- Independent DRAM controllers

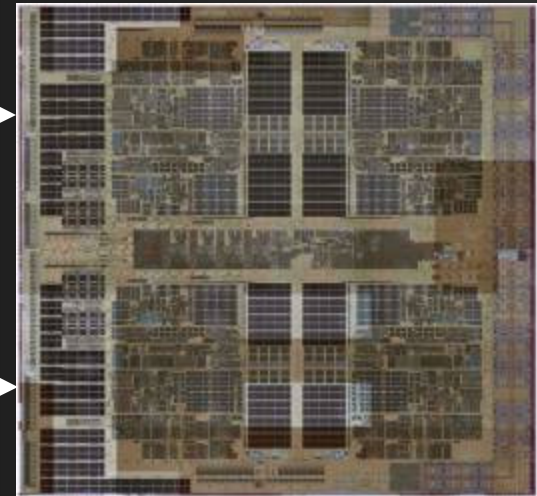
- ▶ Concurrency
- ▶ More DRAM banks
reduces page conflicts
- ▶ Longer burst length
improves command
efficiency

- Optimized DRAM paging
- Re-architect NB for higher BW
- Write bursting
- DRAM prefetcher
- Core prefetchers

DDR module



DDR module



Delivering more DRAM bandwidth

enhancements in AMD Barcelona processor

- Independent DRAM controllers
- **Optimized DRAM paging**
 - ▶ Increase page hits, decrease page conflicts
 - ▶ History-based pattern predictor
- Re-architect NB for higher BW
- Write bursting
- DRAM prefetcher
- Core prefetchers

Delivering more DRAM bandwidth

enhancements in AMD Barcelona processor

- Independent DRAM controllers
- Optimized DRAM paging
- **Re-architect NB for higher BW**
 - ▶ Increase buffer sizes
 - ▶ Optimize schedulers
 - ▶ Ready to support future DRAM technologies
- Write bursting
- DRAM prefetcher
- Core prefetchers

Delivering more DRAM bandwidth

enhancements in AMD Barcelona processor

- Independent DRAM controllers
- Optimized DRAM paging
- Re-architect NB for higher BW

- **Write bursting**

► **Minimize Rd/Wr Turnaround**

- DRAM prefetcher
- Core prefetchers

Delivering more DRAM bandwidth

enhancements in AMD Barcelona processor

- Independent DRAM controllers
- Optimized DRAM paging
- Re-architect NB for higher BW
- Write bursting
- **DRAM prefetcher**
 - ▶ Track positive and negative, unit and non-unit strides
 - ▶ Dedicated buffer for prefetched data
 - ▶ Aggressively fill idle DRAM cycles
- Core prefetchers

Delivering more DRAM bandwidth

enhancements in AMD Barcelona processor

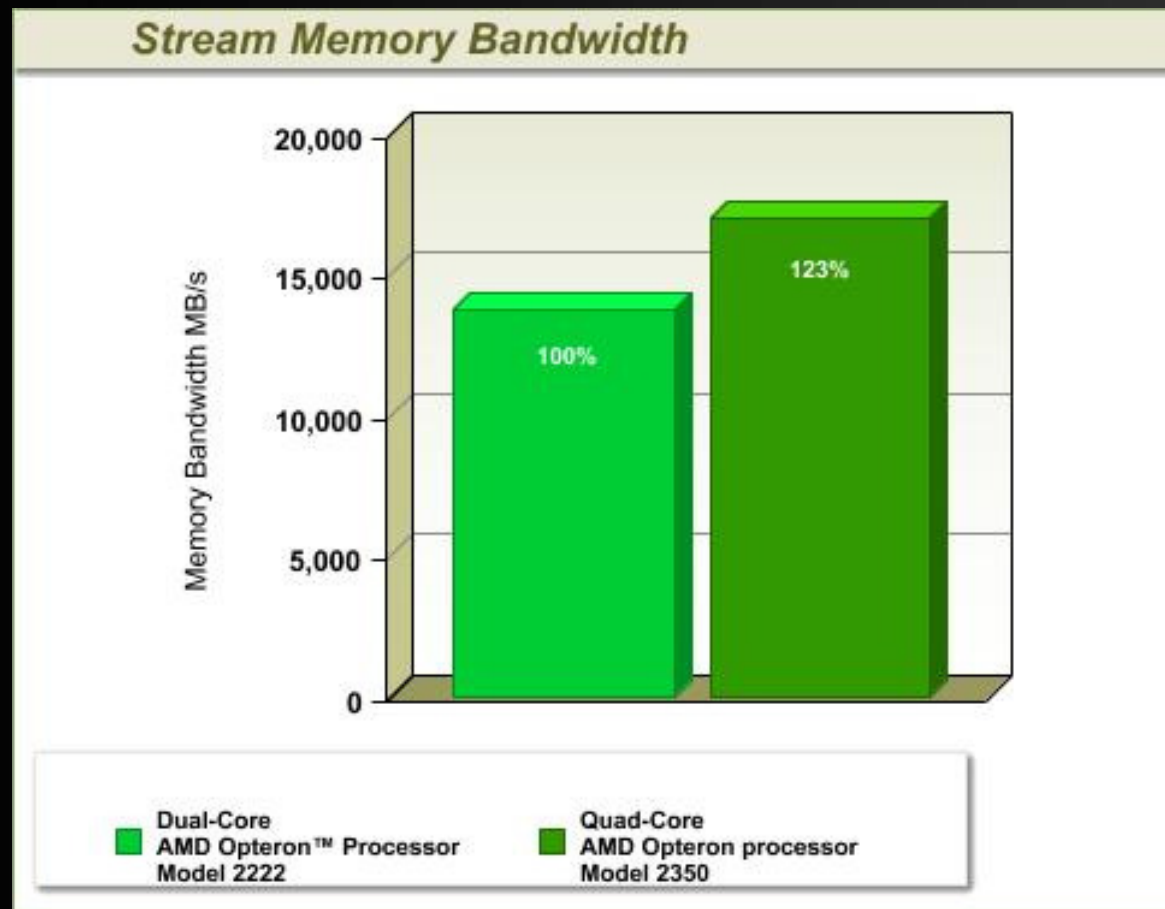
- Independent DRAM controllers
- Optimized DRAM paging
- Re-architect NB for higher BW
- Write bursting
- DRAM prefetcher

- Core prefetchers

- ▶ DC Prefetcher fills directly to L1 Cache
- ▶ IC Prefetcher more flexible
2 outstanding requests to any address

Opteron Memory Bandwidth Evolution

more available bandwidth from the same DDR memory



Balanced, highly efficient cache structure

also a symmetric cache structure

Dedicated L1 data cache

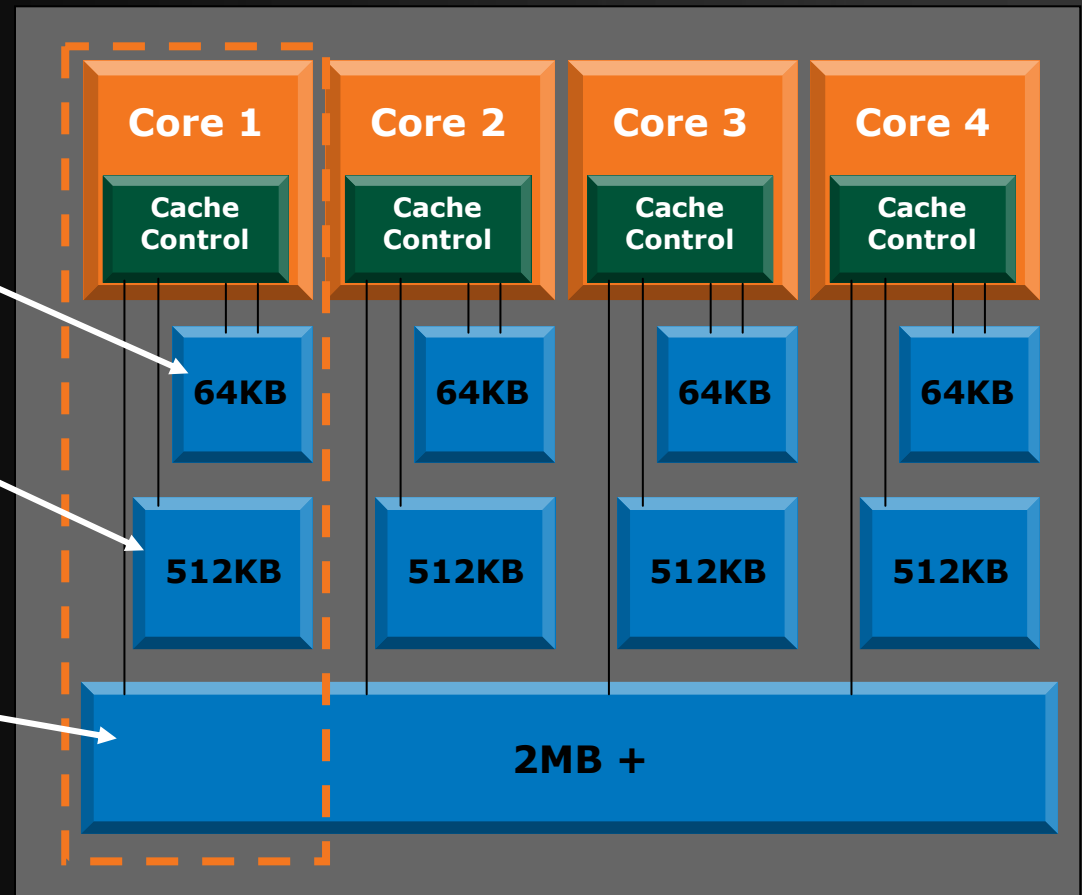
- Locality keeps most critical data in the L1 cache
- Lowest latency
- 2 loads per cycle

Dedicated L2 data/instruction cache

- Sized to accommodate the majority of working sets today
- Dedicated to eliminate conflicts common in shared caches
 - Better for Virtualization

Shared L3 – NEW

- Victim-cache architecture maximizes efficiency of cache hierarchy
- Fills from L3 leave likely shared lines in the L3
- Sharing-aware replacement policy
- Ready for expansion at the right time for customers



The big trend: a lot more cores...

...but relatively only a little more memory performance

- Moore's Law is alive and well, but working in a new way
 - We still get 2x more transistors every 18-24 months...
 - But instead of higher single-thread performance, we get more cores
 - For developers, this changes almost everything
- Multi-threading or multi-processing is essential for performance
 - Threading must become routine, like using printf or "for" loops
 - OpenMP and threaded library code makes it easier
 - But threading is not really the focus of this talk
- Many cores + limited memory bandwidth/latency =
data bottlenecks
 - This talk is about how to avoid or reduce the bottlenecks

What does this all mean, for you?

Now we talk about software!

A useful cross-platform approximation

cache architecture and memory latency

- Every CPU may have different specs for cache
- Different computers have different memory systems
- But we can still make a useful, general approximation:
 - Fastest data access time is L1 cache, let's call it 1x
 - Higher cache levels are 10x
 - Main memory is 100x
- The point is: accessing main memory is very expensive!

#1 Optimize data layout to hide latency

- Latency to access main memory is the most common S_L_O_W operation performed by software
- Idea: reduce the effect of latency by issuing memory requests sooner
- Problem: but the CPU cannot issue a memory request until the address is known
- *How can the CPU generate an address sooner?*

#1 Optimize data layout to hide latency

- Arrays vs. linked lists

Array

Easy random access
Hard to insert/remove data
Sequential addresses

Address can be *calculated*

Linked list

No random access
Easy to insert/remove data 😊
Scattered addresses

Address *depends on data*

#1 Optimize data layout to hide latency

traversing an array

list

Address = 100
Address = 200
Address = 300
Address = 400
Address = 500
Address = 600
Address = 700
Address = 800
Address = 900
Address = 1000

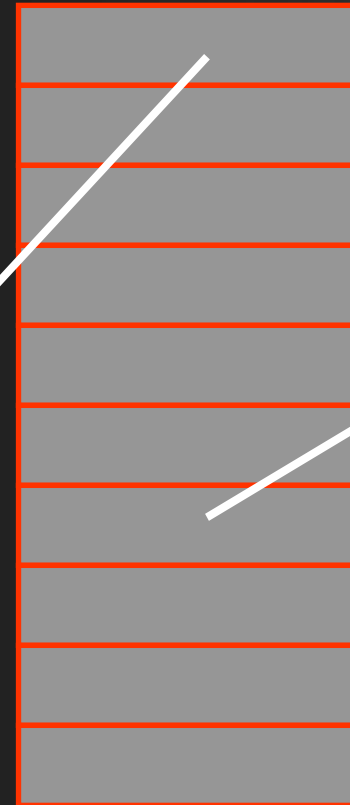


traversing a linked

Address = 100

read
"next"
pointer

Address = 700

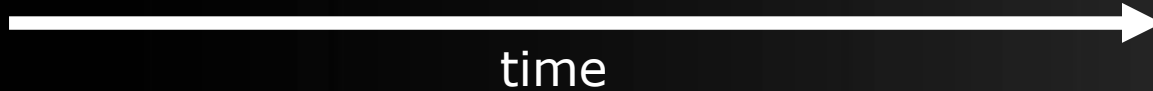


#1 Optimize data layout to hide latency

traversing an array:

Fetch address 100
Fetch address 200
Fetch address 300
Fetch address 400
Fetch address 500

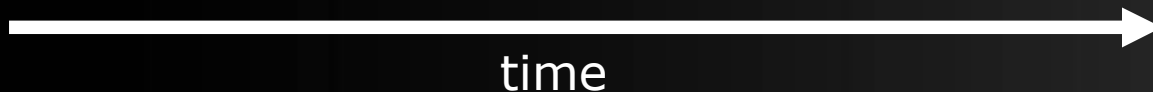
*Memory requests overlap!
This is a form of ILP
(Instruction-Level Parallelism)*



traversing a linked list:

Memory requests cannot overlap!

Fetch address 100... wait for "next" data from memory... Fetch address 700...





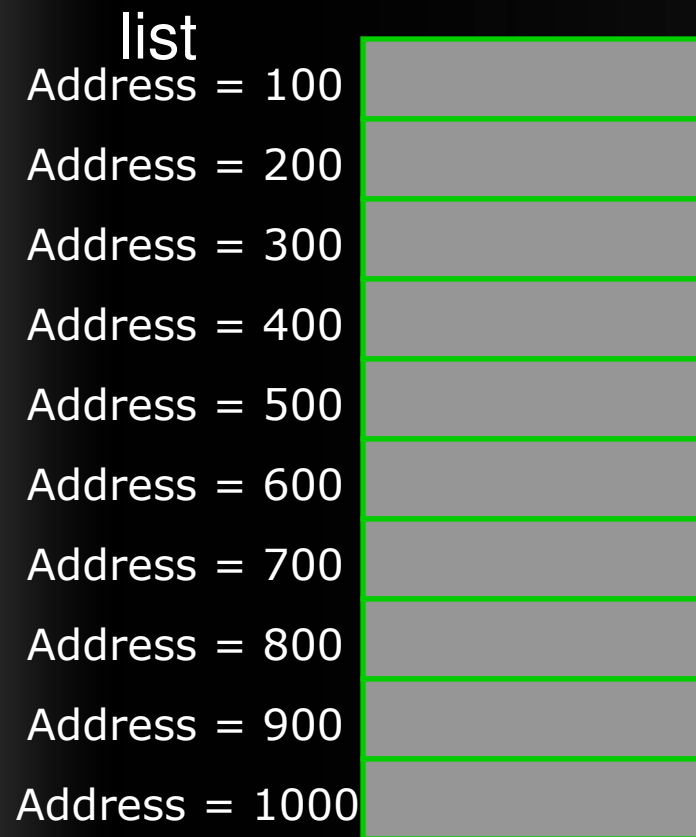
Demo

Array and linked list example in Visual Studio

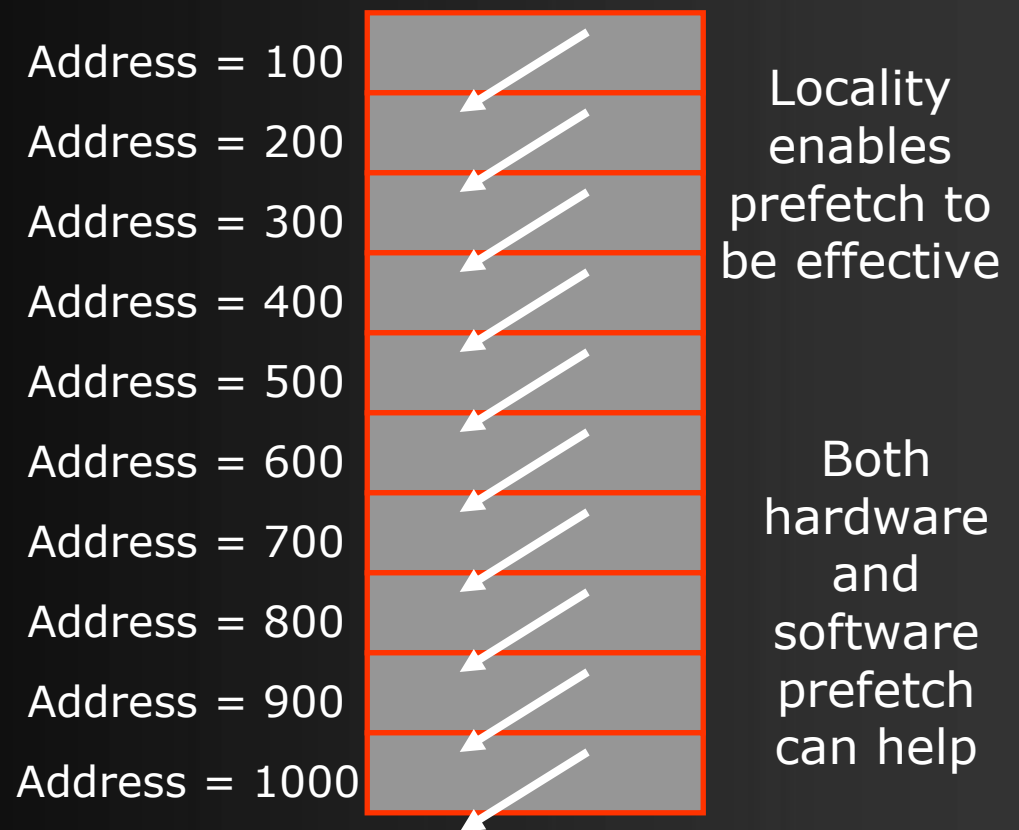
#1 Optimize data layout to hide latency

get higher performance from linked lists

traversing an array



traversing a *sorted* linked

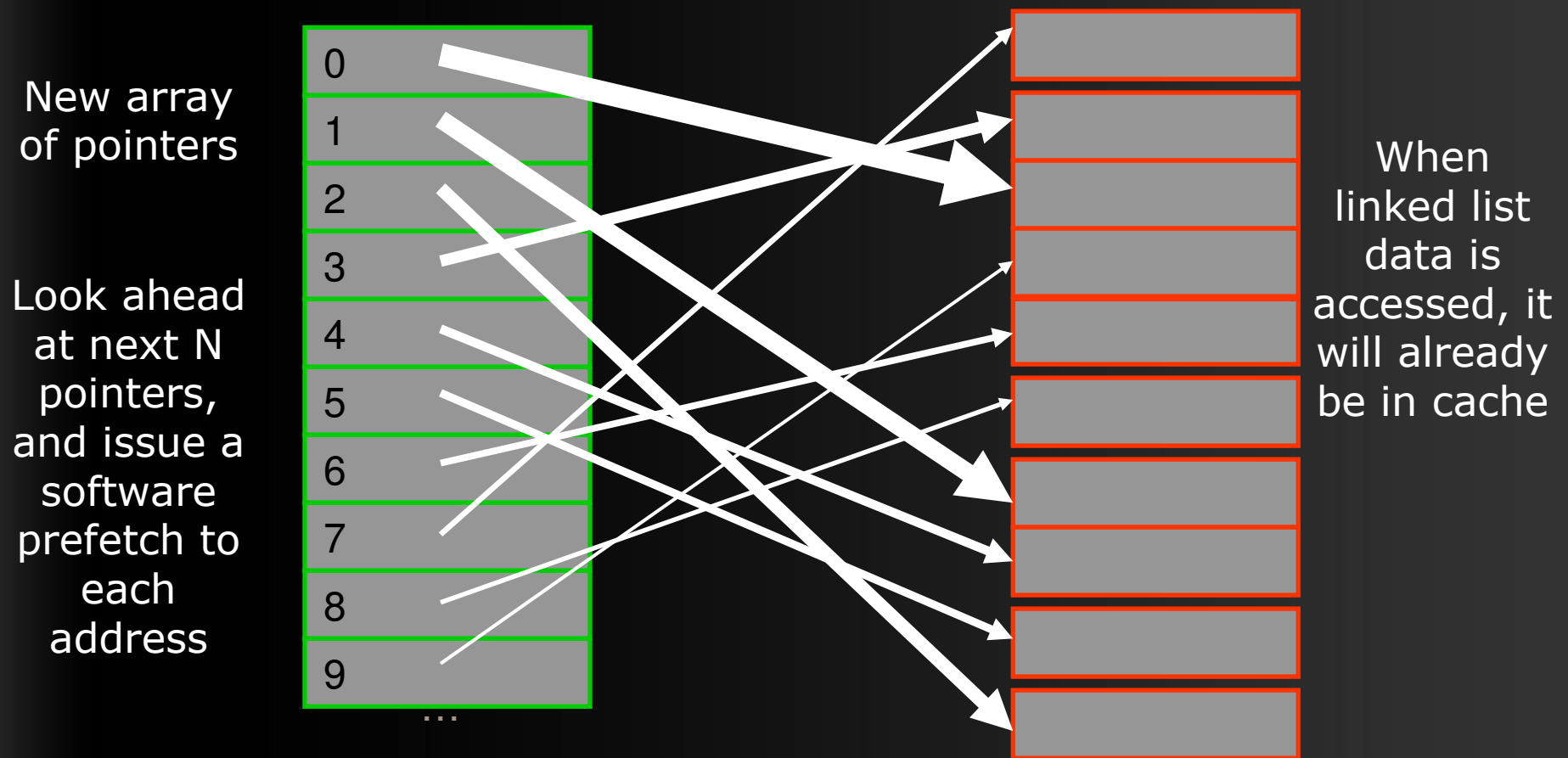


Sometimes it's beneficial to look at the actual *value* of an address pointer

#1 Optimize data layout to hide latency

get higher performance from linked lists

Another trick: build a dynamic index array to guide software prefetch



This method requires relatively small changes to your code ☺

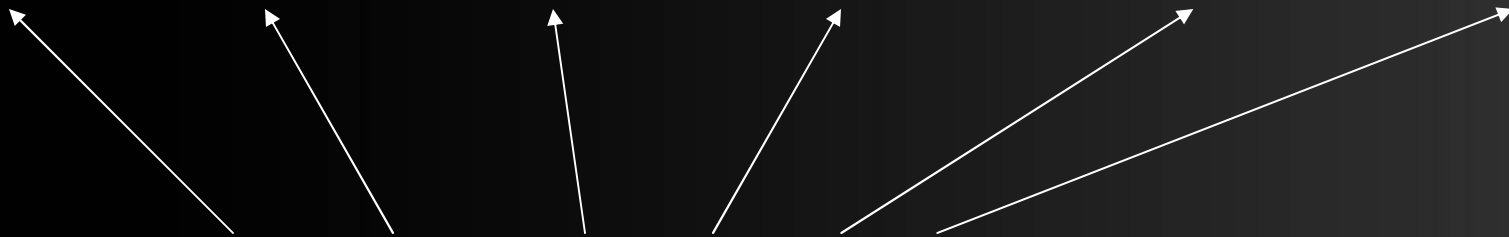
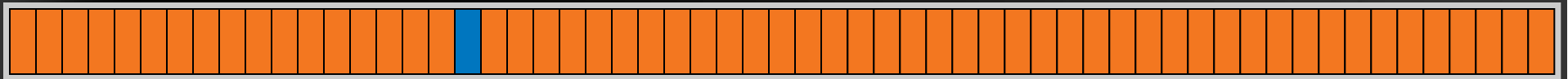
#1a A word about the TLB

TLB is the Translation Lookaside Buffer

- Windows® uses Virtual Memory
- Every address must be translated from virtual to physical
- This translation mechanism is highly optimized: TLB
- TLB is just like a cache, but it stores memory page addresses
 - Typical page size is 4K, larger pages are supported in CPU and some OS's
- TLB has a finite size
 - On Barcelona, L1 TLB has 48 entries, L2 has 512 entries
- Just remember this:
- *Locality of data helps the TLB work more effectively*
- Fewer TLB misses = higher performance
- More important with larger data sets

#2 Optimize data layout for cache

- A cache line is the “atomic unit” of storage
 - Read **one byte** from memory...
 - ...and you get **64 bytes** in your cache!



- Make sure the other 63 bytes are useful !
- *Maximize data locality*

#2 Optimize data layout for cache

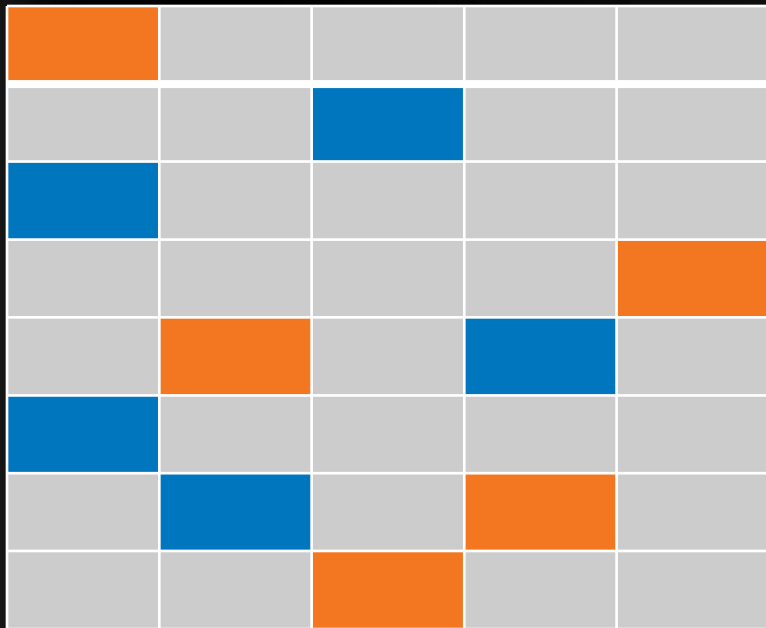
- Dense packing of data in struct and class members
 - Use the smallest appropriate data type
 - Use byte instead of int to store a small integer
 - Use a small index value (byte/short/int) instead of a pointer when possible
 - Beware: the compiler may use padding to align elements!
 - Use sizeof() to check the actual size in memory

#2 Optimize data layout for cache

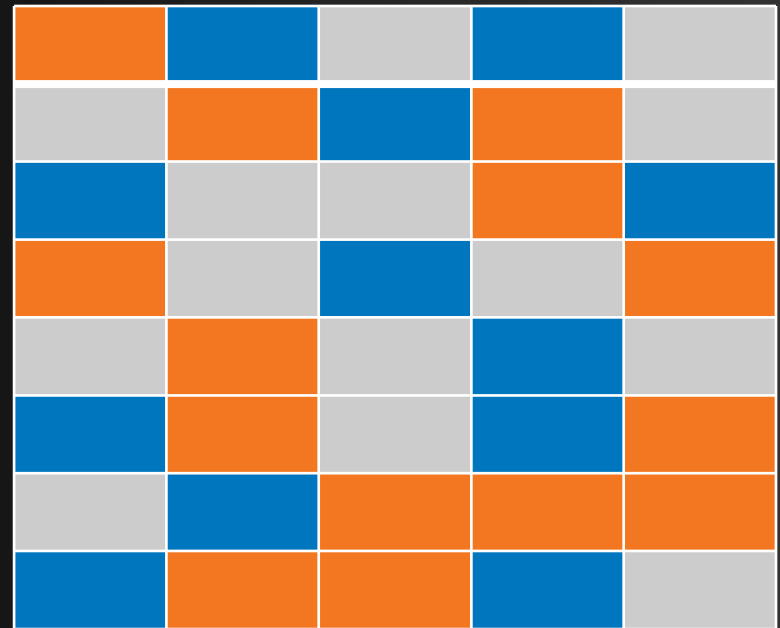
- Consider splitting struct into “hot” and “cold” parts
 - Hot part contains all the frequently used elements
 - Cold part stores elements that are used less often
 - Based on *dynamic* behavior with *real workloads*
 - You really should profile your code, to learn this

#2 Optimize data layout for cache

Low utilization of cache data



Dense packing of useful data



More useful data per cache line = fewer memory requests = higher performance 😊

#2 Optimize data layout for cache

considerations for the new level 3 cache

- Share what's in Level 3 cache, between threads
 - Symmetric multi-core access to Barcelona's L3 cache
 - *Shared data stays in L3 longer than non-shared data*
 - L3 is ideal for producer/consumer multi-threading models
 - L3 will also tend to store code, if you have a large code set
 - Run the same code on multiple threads
 - Data-parallel threading, e.g. OpenMP loops

#3 Cache management: avoid re-loading

cache blocking of data: increase locality

- Simple non-block way: probably not optimal for large data sets
- It uses more memory bandwidth than necessary, re-loading data

```
For each data item, do process A; // read memory
For each data item, do process B; // read memory again
For each data item, do process C; // read memory again
...
```

- Process *blocks* of data in-cache: only read memory once!

```
Divide data into cache-friendly blocks of N items
For each block of N data items (a few Kbytes)
{
    Do process A; // read memory only once
    Do process B; // data still in cache!
    Do process C; // data still in cache!
    ...
}
```

#3 Cache management: non-temporal data

- Avoid filling the cache with data you don't need again
 - Non-temporal data means “don't need it again soon”
 - Only keep cache data that has valuable *temporal locality*
- Explicit cache control can help performance
 - Non-temporal prefetch
 - Streaming store

#3 Cache management: non-temporal data

- Loading non-temporal data: prefetchNTA instruction
 - Non-temporal SW prefetch reads data into L1 cache, of course
 - Like any prefetch, it can help hide memory latency
 - • Will *not evict* to L2 later; data just gets over-written in L1
 - Execute one prefetch per 64-byte cache line, ideally
 - Implemented as a compiler intrinsic function in Visual Studio
 - Easy to use! No ASM code required.
 - `_mm_prefetch(char* ptr, _MM_HINT_NTA);`

#3 Cache management: non-temporal data

- The other way data gets into cache is by executing store instructions, e.g. when your code writes to “memory”
- You are actually writing to the cache, not to memory
- And nearby memory data must be *read* into cache first
 - Remember: a cache line is an atomic 64-byte unit
 - It gets filled from memory when allocated

#3 Cache management: non-temporal data

- Storing non-temporal data: `movNTxx` instructions
 - Does not allocate a cache line, uses a 64-byte *write-combine buffer*
 - Store a complete SSE register, packed `int` or `fp` data
 - `_mm_stream_ps(float* ptr, __m128 a); // 16 bytes`
- Scalar streaming store: new in AMD Phenom™ processor and Visual Studio 2008!
- No awkward data packing required
 - `_mm_stream_ss(float* ptr, __m128 a); // 4 bytes`
 - `_mm_stream_sd(double* ptr, __m128 a); // 8 bytes`

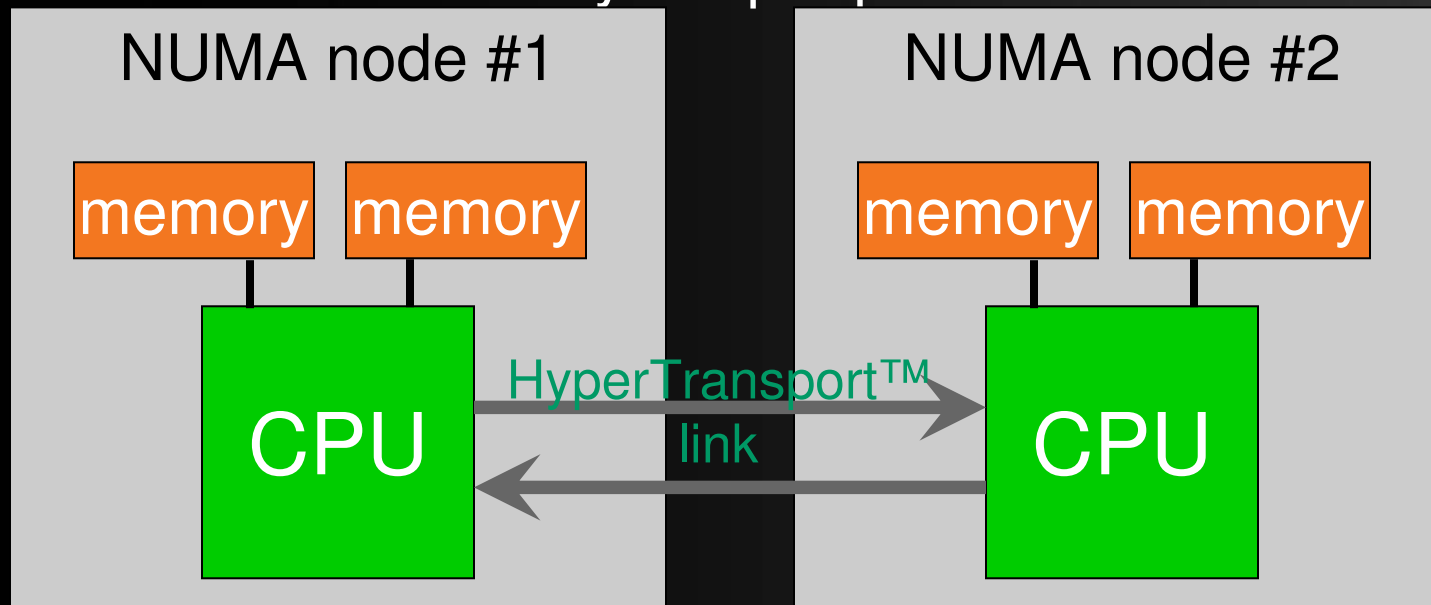
#3a A word about instruction cache

- A large working set of code can stress I-cache
 - Compiling for small code might give the highest performance!
 - Try compiling with /O1
 - Sometimes /O2 or /Ox are not the fastest
- Also try Whole Program Optimization
 - /GL and /LTCG
 - And try profile guided optimization
- Good D-cache (data cache) management can also reduce I-cache pressure, because L2 and L3 store both instructions and data

#4 NUMA and memory affinity

NUMA = Non Uniform Memory Access

- Multi-socket AMD Opteron™ machines are NUMA architectures, which is the industry trend
 - Multiple memory controllers = higher total system bandwidth
 - Local memory bank has somewhat lower *latency*
 - If every thread/process uses local bank, more *bandwidth* is available
- OS can automatically keep a process on a NUMA node



#4 NUMA and memory affinity

NUMA = Non Uniform Memory Access

- Multi-threaded applications should be NUMA-aware!
- GetLogicalProcessorInformation API in Windows
 - Identify which logical processors are in which NUMA node
 - Also reveals cache sharing and other details
- SetThreadAffinityMask can *restrict a thread* to certain NUMA node(s) or logical processors
- *Allocate memory with affinity* by calling allocation function from the desired NUMA node
- Also GetNumaHighestNodeNumber and other functions
- See MSDN for all the details on NUMA API functions

#5 AMD CodeAnalyst Profiler

- Optimization requires real, measured performance data
- Easily see what your code is actually doing
 - No modification to your code is necessary
 - View performance data for *all processes* that are running
- Timer based sampling
 - The classic “find the hot spots” analysis
- Event based sampling
 - L1 cache miss, L2 cache miss, I-cache miss, branch mispredict, misaligned memory access, TLB miss, etc.
- Download AMD CodeAnalyst from developer.amd.com (you need to login first)
- Read the easy tutorial under “help” menu

Homework assignment!

- Profile your code using CodeAnalyst
- See what are the “hot spots”
 - Are you surprised by what you see?

For extra credit:
sort your linked list
by address order,
and try prefetchNTA
😊

If your code uses linked lists:

- Identify which data member(s) get accessed in hot spots
- Be sure to look for ‘cache miss’ and ‘TLB miss’ events
- Re-order struct to place “hot” members near “next” link

```
struct foo {
```

- `foo* next;`
- `char name[48];`
- `BOOL hot_variable;`

Bad! The “hot”
members are separated

```
struct foo {
```

- `foo* next;`
- `BOOL hot_variable;`
- `char name[48];`

Good! The “hot” members
are together, and can ride
on the same cache line...
faster!

Related Content

Visit developer.amd.com

- [Software Optimization Guide for Barcelona “Family 10h”](#)
- [Bios and Kernel Developer’s Guide for “Family 10h”](#)
- Optimization white paper in Windows section:
 - [“Performance Optimization of 64-bit Windows Applications for AMD Athlon™ 64 and AMD Opteron™ Processors using Microsoft Visual Studio 2005”](#) (covers 32-bit and 64-bit)
 - Other papers on NUMA, compilers, virtualization, etc.
- Ben Sander’s Barcelona slides from Microprocessor Forum on developer.amd.com



New “Barcelona”
Software Developer
Resources Page




*Family 10h = AMD Barcelona CPUs

mike.wall@amd.com

Trademark Attribution

AMD, the AMD Arrow logo, AMD Athlon, AMD Opteron and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Windows is a trademark of Microsoft Corporation in the United States and/or other jurisdictions. HyperTransport is a licensed trademark of the HyperTransport Technology Consortium. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.



Visit
our
booth
at

Game Developers'
Conference

08

Learn
Network
Inspire

February
18-22
San Francisco
Moscone Center

gdconf.com

The banner features several circular inset images: a man in a blue shirt holding a trophy, a group of people at a conference table, and a person at a computer monitor. The design is accented with large, overlapping pink and white circles.